**Goal:** learn how to build client/server application that communicate using sockets

**Socket API**

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

A **socket** is a host-local, application-created, OS-controlled interface (a “door”) into which application process can both send and receive messages to/from another application process.
**Socket:** a door between application process and end-end-transport protocol (UCP or TCP)

**TCP service:** reliable transfer of **bytes** from one process to another
Socket programming with TCP

Client must contact server
• server process must first be running
• server must have created socket (door) that welcomes client’s contact

Client contacts server by:
• creating client-local TCP socket
• specifying IP address, port number of server process
• When client creates

When contacted by client, server TCP creates new socket for server process to communicate with client

allows server to talk with multiple clients
source port numbers used to distinguish clients

TCP provides reliable, in-order transfer of bytes (“pipe”) between client and server
Server (running on hostid)

create socket, port=x, for incoming request:
welcomeSocket = ServerSocket()

wait for incoming connection request
connectionSocket = welcomeSocket.accept()

read request from connectionSocket
write reply to connectionSocket
close connectionSocket

Client

create socket, connect to hostid, port=x
clientSocket = Socket()

send request using clientSocket
read reply from clientSocket
close clientSocket
Stream jargon

• A **stream** is a sequence of characters that flow into or out of a process.

• An **input stream** is attached to some input source for the process, e.g., keyboard or socket.

• An **output stream** is attached to an output source, e.g., monitor or socket.
Example: Java client (TCP)

```java
class TCPClient {
    public static void main(String argv[]) throws Exception {
        String sentence;
        String modifiedSentence;
        BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
        Socket clientSocket = new Socket("hostname", 6789);
        DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());
        Create input stream
        Create client socket, connect to server
        Create output stream attached to socket
        BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
        Socket clientSocket = new Socket("hostname", 6789);
        DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());
    }
}
```
```
BufferedReader inFromServer =
    new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();
outToServer.writeBytes(sentence + '\n');
modifiedSentence = inFromServer.readLine();
System.out.println("FROM SERVER: " + modifiedSentence);
clientSocket.close();
```
Example: Java server (TCP)

```java
import java.io.*;
import java.net.*;

class TCPServer {
    public static void main(String argv[]) throws Exception {
        String clientSentence;
        String capitalizedSentence;
        ServerSocket welcomeSocket = new ServerSocket(6789);
        while(true) {
            Socket connectionSocket = welcomeSocket.accept();
            BufferedReader inFromClient =
                new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
            
            Socket connectionSocket = welcomeSocket.accept();
            BufferedReader inFromClient =
                new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
        }
    }
}
```
Example: Java server (TCP), cont

Create output stream, attached to socket

DataOutputStream outToClient =
    new DataOutputStream(connectionSocket.getOutputStream());

Read in line from socket

clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';

Write out line to socket

outToClient.writeBytes(capitalizedSentence);

End of while loop, loop back and wait for another client connection
Socket programming with UDP

UDP: no “connection” between client and server

• no handshaking
• sender explicitly attaches IP address and port of destination to each packet
• server must extract IP address, port of sender from received packet

UDP: transmitted data may be received out of order, or lost

application viewpoint

UDP provides unreliable transfer of groups of bytes (“datagrams”) between client and server
Client/server socket interaction: UDP

**Server (running on host id)**

- create socket, port = x.
  - serverSocket = DatagramSocket()
- read datagram from serverSocket
- write reply to serverSocket specifying client address, port number

**Client**

- create socket, clientSocket = DatagramSocket()
- Create datagram with server IP and port=x; send datagram via clientSocket
- read datagram from clientSocket
- close clientSocket
Example: Java client (UDP)

Client process

Output: sends packet (recall that TCP sent "byte stream")

Input: receives packet (recall that TCP received "byte stream")

Client UDP socket

From network

UDP packet

ReceivePacket

To network

UDP packet

SendPacket

InputStream

Keyboard

monitor

InputFromUser

client UDP socket
Example: Java client (UDP)

```java
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception {
        BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress IPAddress = InetAddress.getByName("hostname");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];

        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
```

Example: Java client (UDP), cont.

```java
Create datagram with data-to-send, length, IP addr, port

DatagramPacket sendPacket =
new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

clientSocket.send(sendPacket);

Send datagram to server

DatagramPacket receivePacket =
new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

Read datagram from server

String modifiedSentence =
new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
```
Example: Java server (UDP)

```java
import java.io.*;
import java.net.*;

class UDPServer {
    public static void main(String args[]) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];

        while(true) {
            DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
        }
    }
}
```
String sentence = new String(receivePacket.getData());
InetAddress IPAddress = receivePacket.getAddress();
int port = receivePacket.getPort();
String capitalizedSentence = sentence.toUpperCase();
sendData = capitalizedSentence.getBytes();
DatagramPacket sendPacket =
    new DatagramPacket(sendData, sendData.length, IPAddress, port);
serverSocket.send(sendPacket);

End of while loop, loop back and wait for another datagram